Enabling Resource Scheduling in Cloud Distributed Videoconferencing Systems

Álvaro Alonso, Pedro Rodríguez, Ignacio Aguado, Joaquín Salvachúa
Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
Madrid, Spain
email:{aalonsog, prodriguez, iaguado, jsalvachua}@dit.upm.es

***Abstract*—When deploying videoconferencing systems in Cloud based infrastructures, one of the most complex challenges is to distribute Multipoint Control Units (MCUs) among different servers. By addressing this challenge, we can improve the flexibility and the performance of this type of systems. However, to actually take advantage of the Cloud possibilities we have also to introduce mechanisms to dynamically schedule the distribution of the MCUs across the available resources. In this paper, we propose a resource scheduling model for videoconferencing systems and, starting from an existing MCU distribution architecture, we design a solution to enable the resource scheduling basing on custom criteria. These criteria can be based on the characteristics of each server or in their status in real time. We validate the extended model by setting up a typical videoconferencing deployment among a set of Cloud providers and testing a decision algorithm. We conclude that the proposed model enables the use of a wide range of algorithms that can be adapted to the needs of different Cloud deployments.**

***Keywords*—cloud computing; videoconferencing; distributed MCU; scheduling**

I. INTRODUCTION

Nowadays, a very important part of the applications and services we consume over Internet are provided in Cloud [1] infrastructures. The main advantage of using this technology is that one can adapt the amount of provisioned resources for a service based on the demand. In traditional deployments, one has to forecast the demand before obtaining the hardware and this involves a lack of flexibility that can result in a waste of resources. However, using Cloud based deployments one can dynamically scale a service in almost real time by adding or removing computing capacity. In other words, Cloud Computing provides the illusion of infinite computational power on a pay-per-use basis.

In addition to this advantage, this flexibility enables easier and more efficient ways to distribute the services among different servers. Thus, one can balance the system load, replicate instances or geographically distribute them. These operations are done almost instantaneously with a *click* or by calling an API. Even some public Cloud providers offer components that automatically distribute the load between servers. However, these mechanisms are usually designed to be used by request-response services such as RESTful or web services.

In videoconferencing systems, the communication between participants is usually performed via a central server called Multipoint Control Unit (MCU). MCUs are used to address the signalling and to interchange the media streams between peers. Furthermore, in advanced configurations they are used to record sessions or to transcode video and audio flows.

Today, thanks to technologies such as Adobe Flash [2] and HTML5 [3] with its real-time communications standard Web Real Time Communications (WebRTC) [4], videoconferencing systems are accessible from web applications and mobile devices and its use is open to a higher number of users than ever before. On the other hand, the demand of these services can vary dynamically in short periods of time. The fluctuation in the number of users and the managed sessions results in substantial changes in the computing capacity consumed by the MCUs. Hence, and this is strengthened in [5], deploying MCUs in Cloud infrastructures offers several advantages. One of them is the already mentioned idea of distributing a service among several servers. But, as anticipated, the standard solutions for resource scheduling do not apply to the specific case of MCU distribution.

In traditional web services based on HTTP, it is usually enough with a distribution of the users requests using load balancers. These components are adapted to the requirements of complex web services. However, we argue no general solution for distributed videoconferencing systems is available. Here, the resource use in each of the distributed nodes may depend on several parameters besides the number of users. This can be understood with a very easy example. Suppose an scenario with an MCU distributed in two servers and managing six users each. In the first MCU the six users are connected to the same videoconferencing room and in the second one each user is connected alone to a different room. Obviously, the first MCU is consuming more resources because it is receiving packets from each user and broadcasting them to all of the rest while the second one is only receiving packets from the users without any processing to do. Furthermore, to address the scheduling challenge in this scope we first need to understand which type of criteria we need to take into account and then we must enable a way to schedule the load distribution taking those criteria into account.

In the next Section, we analyse the existing solutions regarding resource scheduling in the Cloud and why they do not cover the videoconferencing scenario. Then, in Section III we describe our scheduling model and extend an existent distributed architecture to cover the needed requirements. Then, in Section IV we validate the solution with a real implementation and a typical use case. Finally, in Section V we enumerate the main conclusions obtained and we analyse the different research lines we open to continue the work.

II. RELATED WORK

Scheduling Cloud Computing resources has always been a subject under study. Being able to dynamically adapt and change the available resources depending on the demand of the users is a key point in services deployed in the Cloud. Many strategies have been designed in order to solve this

 ISBN: 978-1-61208-460-2

problem, using techniques like the Genetic Algorithm [6], Particle Swarm Optimization (PSO) [7], Ant Colony Optimization (ACO) [8] or Load Balancing Ant Colony Optimization (LBACO) [9]. All of them try to find an optimal resource allocation for workload in a generic Cloud architecture.

In spite of all of this work, there is not any solution specifically designed for a videoconferencing service. However, there exists a model which, if properly developed, could be useful. The model described in [10] could be the first step of Scheduling a videoconference service in the Cloud. There, the authors propose an architecture that allows to easily divide an MCU in atomic parts called One To Many (OTMs). An OTM is a software component that basically broadcasts a video/audio stream to many participants. And these OTMs can be distributed among different servers without introducing extra latency in the communications.

It is important to define a valid and efficient algorithm for this kind of services to obtain better performance and more flexibility. However, in order to achieve this goal it is essential to propose a mechanism that allows us to implement it in the model described above. No other solutions have been defined for this new layer of functionality in a videoconference service, but this work defines, implements and tests one.

We extend the OTM model by adding the layer that allows us to decide how to schedule the resources among the distributed OTMs (dOTMs). These decisions are customisable and can be based on the characteristics or the status of the servers where the OTMs are deployed. As an example case we also provide a basic algorithm. With this mechanism, specifically designed for a videoconferencing service architecture, it is possible to define new algorithms, grouping different criteria.

III. VIDEOCONFERENCING RESOURCE SCHEDULING

We have concluded that the first thing we need to start working for an efficient videoconferencing services distribution is to design a model that allows us to introduce custom decisions to schedule the resources. In this section, we describe the model we propose by analysing the main characteristics it has to cover. We also design an architecture that extends the existing MCU distribution model dOTMs to comply with those requirements.

A. Model description

As introduced above, distributed MCU servers differ from common web services in many aspects. Videoconferencing is not a request-response service and the amount of resources needed per user depends on the size of the videoconferencing sessions. According to this, we cannot distribute the users homogeneously among distributed MCUs. When connecting a new user, we need to know certain information about the current status of each of the available servers to decide which to use. Once the decision is taken, we need to be able to assign the connection to the selected MCU. Thus, the model we are proposing needs to cover three main aspects:

1) Decision layer: Having a distributed MCU, we need a central component in charge of deciding where to allocate each connection. This component has to be configurable with custom decision algorithms that take as input the list of available MCUs and their information and return the selected one. Once the decision is taken this component has to be able to communicate directly with the selected MCU to allocate the resources there. Moreover, the algorithms can be automatically modified in real-time taking into account the feedback received from the MCUs.

2) MCU registration: To be able to communicate with a specific MCU, the decision layer must have a list with all the available ones. So, an MCU has to be registered in the central component when it is added to the pool of available MCUs. In the registration, the MCU can specify a set of fixed characteristics of itself that can be used by the algorithms at decision time.

3) MCU report: The fixed information set at registration time is not enough. In the decision layer we also need real-time information about the status of the MCUs. Thus, the algorithms can decide also basing on parameters such us the CPU or Memory use of each MCU. Therefore, we also need a returning channel between the decision layer and each MCU.

B. Architecture design

To put these three requirements in a real MCU distribution model we have designed an architecture that extends the one described in [10]. The key of this architecture is the design of a mechanism to split a traditional software MCU into smaller components called OneToManys (OTMs). A OTM receives packets from a source and forwards it to many destinations, usually participants in a videoconferencing session. If all the participants of a session are sharing their media with the others, the MCU receives the media packets from each participant and forwards them to the rest. Thus, with the dOTMs model, we can manage a session using a OTM for each participant. Running each OTM in a single process we can distribute the same session among different servers. To achieve this it is necessary to isolate the media layer of the MCU.

The dOTMs model proposes the division of a traditional MCU into three layers: signalling, control and media. Therefore, as detailed in [11], this separation results in an architecture with three main components:

- *OTM*: the software unit in charge of receiving media packets from a participant and broadcast it to many.
- *Agent*: the component in charge of managing OTMs. We have an Agent per each machine in which we want to host OTMs.
- *Controller*: manages videoconference rooms and interchanges the signalling messages between clients and OTMs. It also communicates with Agents to start and stop OTMs.

To perform the signalling, these components communicate between them using a Message Bus. When a new participant joins a videoconferencing session and wants to publish media, the Controller asks an Agent to create a new OTM and establishes the signalling between the client and the created OTM. When they are connected the media communication takes place directly between them. When other participant wants to subscribe to the published media, the Controller searches the corresponding OTM and establishes the connection in the same way.

If we start Agents in different servers we can distribute even the same videoconferencing session in different infrastructures taking advantage of the Cloud benefits exposed before. However, the dOTMs model does not specify how to schedule the resources between the available Agents. We have to adapt the model to the requirements explained above modifying the way

 ISBN: 978-1-61208-460-2

in which the Controller and the Agents communicate between them.

Reviewing the three requirements explained in the model description and analysing the current dOTMs architecture we can observe the following:

1) Agent decision: When a new OTM is required in a session, the Controller selects the Agent in round-robin mode sending the creation request to one of the available Agents each time. We need to support the creation of OTMs in specific Agents basing on custom algorithms. In order to enable that, we have to introduce a way to allow the Controller to communicate directly with a specific Agent.

2) Agent registration: The Controller does not really have awareness about the existing Agents. To communicate with them sends a message to a message bus and the bus is the one that redirects it to one of the subscribed Agents. We need to add a mechanism to provide the Controller a list of the available Agents in every moment. In other words, every Agent has to be registered in the Controller.

3) Agent report: To take the decision of which Agent select to create a new OTM, the Controller needs information of the available Agents. It is specially interesting to have information of the status of each Agent in real time. In the current configuration, once an OTM is created the Controller sends messages directly to it and does not need to communicate with the Agent anymore. To support a real time report from the Agents we need to enable a persistent communication channel between each Agent and the Controller.

Figure 1 illustrates the extensions we propose in this paper. We introduce a new message queue for each of the Agents (*Agent id*). These queues are configured in a direct unicast mode and used by the Controller to communicate directly with an Agent. When a new Agent is added to the environment it creates a new queue to be able to receive messages. All the Agents are still subscribed to the common queue (*Agent*) but now that queue is able to send broadcast messages to all the Agents. This queue is used by the Controller to send a periodic message that will be answered by the existing Agents. In the response of these messages each Agent includes three types of information:

- Contact information: the needed information to contact the Agent. It basically includes the id of the queue created by the Agent.
- Fixed information: constant information that is configured when creating the Agent.
- Realtime information: information about the state of the Agent in each moment.

With this information sent periodically, the Controller has an updated list of the available Agents. Furthermore, when a new OTM is needed it can use the information (both fixed and real-time) of the Agents to decide to which of them delegate the creation. Once decided it sends the creation message using the specific queue. The broadcast queue is still configured to be able to send messages in round robin mode. Thus, if one does not want to specify which Agent to use and wants to evenly distribute the connections between all of them, the system can be used as before. Finally, the periodic broadcast messages are also used as a heartbeat to ensure that an Agent is still available and reachable. The Controller stores a count of the not responded messages to determine when an Agent is not usable anymore.

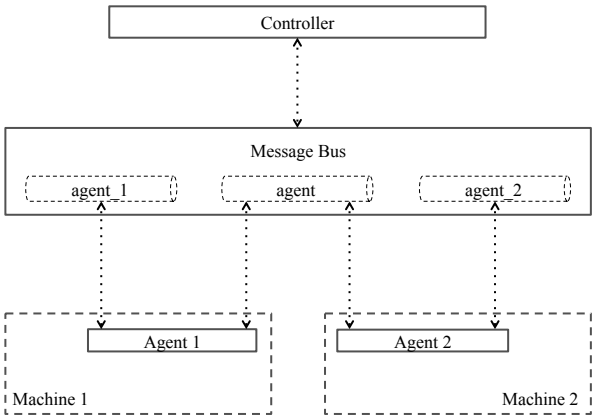

Figure 1: Message Bus configuration

TABLE I: MESSAGES FROM THE CONTROLLER TO THE AGENTS

createOTM	
Requests a new OTM to an Agent. The Agent is selected in round robin mode.	
Queue	Agent
Type	unicast
Parameters	-
Returns	OTM id
createOTM	
Requests a new OTM to a specific Agent.	
Queue	Agent_id
Type	unicast
Parameters	-
Returns	OTM id
deleteOTM	
Forces the destruction of an OTM. The message is sent to all the Agents and only the Agent that owns the OTM deletes it.	
Queue	Agent
Type	broadcast
Parameters	OTM id
Returns	Result code
getAgents	
Requests information to all the Agents available in the system.	
Queue	Agent
Type	broadcast
Parameters	-
Returns	Stats object

In Table I, we can see the detailed specification of the messages we need to enable the proposed mechanism. The two original messages (*createOTM* and *deleteOTM*) are the same, but we need a new *createOTM* message to create in a specific Agent (sent to the unicast queue of the Agent) and the *getAgents* broadcast message to obtain the status of each Agent.

 ISBN: 978-1-61208-460-2

IV. VALIDATION

We have proposed a mechanism that, having a videoconferencing system distributed between several nodes, allows us to dynamically decide how to schedule the load among them. In this section, we validate the mechanism testing a working implementation of the model in a real videoconferencing deployment.

A. Implementation

To validate the solution we use an open source project named Licode [12]. Licode is developed by the authors of this paper and provides a WebRTC compatible videoconferencing system with three main parts.

The first one is an MCU built on top of the WebRTC standard that implements a signalling protocol based on SDP exchange. For the establishment of the media connection it implements the Interactive Connectivity Establishment (ICE) [13] standard and for the encryption of all the media data Secure Real-time Transport Protocol (SRTP) [14] and Datagram Transport Layer Security (DTLS) [15] protocols. The second part is a JavaScript client API that wraps the WebRTC API facilitating the development of videoconferencing applications and adding the necessary modules to communicate with the MCU. Finally, to ensure the security in the signalling between clients and MCU, Licode includes an authorisation module based on Nuve [16]. This module is also in charge of the management of rooms and it is able to start entire MCUs in different machines in order to scale the system.

The current Licode version implements the mechanism exposed in this work using the Advanced Message Queuing Protocol (AMQP) [17] protocol for the message bus, more specifically the RabbitMQ implementation. To send the unicast messages we have implemented an Remote Procedure Call (RPC) mechanism using *direct* exchanges and for the broadcast messages we use *topic* exchanges.

When the Controller sends the *getAgents* broadcast message to get the information about the existing Agents, they respond with the following information:

- *Agent_id*: a unique identifier of the Agent.
- *rpc_id*: the identifier of the AMQP queue to which the Agent is subscribed.
- Metadata object: a JSON object used to store fixed information when starting the Agent.
- Stats object: a JSON object with realtime information about the state of the Agent. It includes CPU and memory usage.
- Keep Alive counter: a counter used to ensure that the Agent is still responding to the requests. When the counter reaches an established limit, the Agent is unregistered in the Controller.

To take the decision of which Agent choose to create a new OTM, the Controller uses the round robin mode as default. Custom policies can be configured by introducing programmatic scripts. A custom script receives as a parameter an object with the Agents information, performs the decision based on it and returns the selected Agent.

B. Experiment description

Using Licode project and its implementation of the model proposed, we have performed an experiment to prove that the solution works in a real use case. We propose a very common scenario in which we (as supposed videoconference as a service provider) need to provide videoconferencing rooms on demand to a variable number of users.

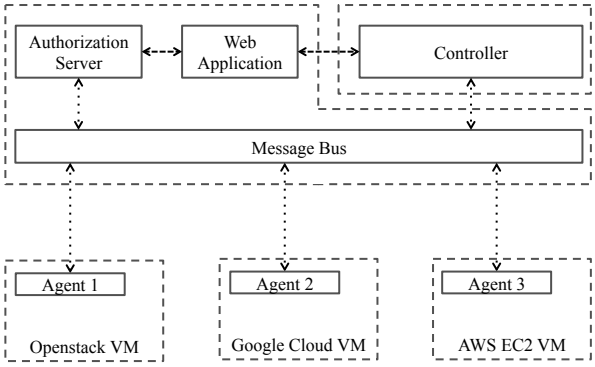

Figure 2: Deployment configuration

TABLE II: INSTANCES USED FOR THE AGENTS

Cloud Platform	Instance Type	CPU	Memory
Openstack	m1.small	1 vCPU	2 GB
Google Cloud	f1-micro	1 vCPU (Shared GCPU)	0.60 GB
Amazon EC2	t2.micro	1 vCPU (Variable ECU)	1 GB

To host the needed resources we own a set of physical computers in which, to facilitate the deployment, we have set up a private cloud environment to deploy virtual machines. If we eventually need more resources we have the possibility of hosting them in two different public cloud providers. The component that we will deploy in the public cloud providers is the Agent in charge of creating new OTMs. In this case the Agent is the bottleneck when it comes to resources use. As seen in [10], dOTMs are limited by CPU, bandwidth or memory are not a limiting factor in public clouds' low processing power instances. Thus, for the purpose of this experiment we will replicate the conditions and simplify load metrics to only take into account CPU measurements.

The deployment of resources in those public Clouds implies an economic cost so the criteria is to prioritise the use of the private Cloud and only when we do not have compute capabilities there, we use the public ones. However, in this case we always want to reserve a part of the computing power of the private Cloud to host private meetings so we will configure a use threshold below the maximum computational level.

Once the set up is ready, we start connecting clients to videoconferencing rooms. First clients will be handled in the private cloud and when the configured threshold is reached, we will start handling the new ones in the public Clouds. To decide which of the two available public Clouds we will select, we will follow load criteria, using in each moment the one that is consuming less resources. We also define a weight factor between both public clouds that can be used to set a priority taking into account different criteria.

C. Deployment set up

To deploy the needed component for the experiment we have chosen Openstack Compute [18] as the private cloud and Amazon Web Services EC2 [19] and Google Compute Engine [20] as the public ones. In Figure 2 we can see a diagram of the configuration.

 ISBN: 978-1-61208-460-2

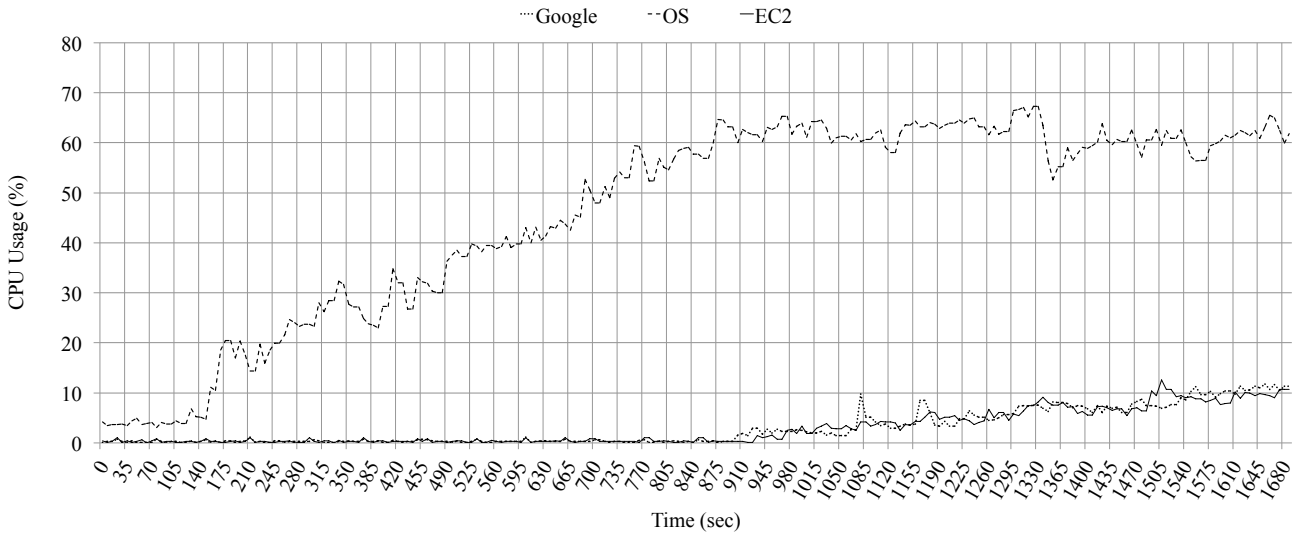

Figure 3: Cloud Providers CPU Usage

Algorithm 1 Agent decision

Require: $cpuThreshold$, $weightFactor$, $osAgent$, $googleAgent$, $ec2Agent$

```
osCPU ← getCurrentCPU(osAgent)
if osCPU < cpuThreshold then
    return osAgent
else
    googleCPU ← getCurrentCPU(googleAgent)
    ec2CPU ← getCurrentCPU(ec2Agent)
    if googleCPU * weightFactor < ec2CPU then
        return googleAgent
    else
        return ec2Agent
    end if
end if
```

We have deployed the Controller in a separate virtual machine. In other virtual machine we have deployed the Authorisation Server (Nuve), the Web Application server and the RabbitMQ server. The infrastructure in which these components have been deployed is not relevant for this experiment. Nor the capabilities of the virtual machines. On the other hand we have deployed an Agent in a virtual machine of each of the mentioned cloud providers, Openstack, Google Cloud and Amazon EC2. The characteristics of the selected virtual machines are described in Table II. The operating system used is Ubuntu 14.04 LTS for all of them.

The small size and computing power of the selected virtual machines is not a problem in the scope of the scenarios we designed. It is very convenient to have a low enough processing power that we can saturate easily to test the ability to allocate OTMs. Regarding the amount of bandwidth available for this type of instance, we have tested it is enough to handle the amount of users we are going to connect.

To host the clients we have used the same instance type as the Amazon EC2 Agent. They connect using Chromium [21] browser (the open-source project behind the Google Chrome) version 45.

Algorithm 1 shows the decision policy we have configured in the Controller. The algorithm is executed everytime we need a new OTM. In the Metadata object of each Agent, we can find the cloud provider where it is running. This way we obtain the values of $osAgent$, $googleAgent$ and $ec2Agent$. We set this parameter in each Agent at boot time. On the other hand, the Stats object allows as to get the current CPU consumption of each Agent ($getCurrentCPU()$ method). In this experiment we have configured the following constants:

- $cpuThreshold$: 60 %
- $weightFactor$: 1

With this threshold value we ensure that a 40% of the private cloud computing capability is reserved for special rooms that we need to host internally. In this experiment we have set a neutral weight factor. This means that there is no priority between the public cloud providers. When the public part is needed, we always create the Agent in the infrastructure that is consuming less CPU. However, using this factor we could design algorithms that establish priorities taking into account advanced criteria such us the pricing, the geographical location, etc. Furthermore, we can use algorithms with feedback that modify the value of that factor taking into account real-time aspects, such as the fluctuation of the pricing.

D. Results

We have created six videoconferencing rooms and connected clients to a random one every 35 seconds, starting in second number 110. This dynamic is only changed in second number 1320, when five clients disconnect from their rooms. The results of the experiment are showed in Figure 3.

As it can be observed, on one hand the CPU usage in the Openstack vitual machine keeps growing as new clients connect to the service. On the other hand, the activity in the Google Cloud and Amazon EC2 ones is almost null. The first key point can be found at second number 875, where the threshold of 60% is first exceeded. From that moment the CPU Usage in Google Cloud and Amazon EC2 virtual machines starts growing, while the Openstack one suffers minor variations but remains almost constant. The growth of

 ISBN: 978-1-61208-460-2

Google Cloud and Amazon EC2 stops at second 1330 as a consequence of the disconnection of five clients explained before. From that moment, the CPU usage of the Openstack virtual machine stops exceeding the threshold, so new clients start connecting again to it instead of connecting to Google Cloud or Amazon EC2 ones. Finally, in second number 1410 the threshold is exceeded again, so the Openstack CPU usage remains constant and the Google Cloud and Amazon EC2 starts growing again until the end of the measures in second 1680.

V. Conclusions and future work

Cloud Computing provides numerous benefits to scalable and distributed services. There are several studies regarding how to efficiently distribute web services or databases taking advantage of the Cloud. But in relation with videoconferencing systems and MCU servers the literature is not so extensive. This type of systems has particular characteristics that make the traditional load balancers not optimal.

In this paper, we have defined a model for scheduling resources in videoconferencing Cloud deployments and designed an architecture that enables this scheduling basing on the realtime status of the MCUs that are managing the sessions. Thanks to this new model, we can design cloud-based scenarios in which we distribute the load according to advanced decision algorithms. To validate that the model actually covers the requirements of videoconferencing scalable systems, we have set up a real deployment using a set of well known Cloud providers and illustrating a very common use case. For this we have used a complete implementation of the model configuring it with an algorithm prototype that simulates the requirements of a videoconferencing as a service provider.

The main conclusion after the experiment is that the proposed model actually works and offers a customisable way to distribute a videoconferencing service according to the specific requirements of each deployment. Once we have the tools to enable this custom decisions, the main challenge is to study algorithms that could standardise the requirements of typical videoconferencing scenarios. Thus, multiple lines of research are opened from this point. The immediate one is to test other kind of algorithms that take into account different criteria than the load of the virtual machines. For instance, a scheme based on the pricing of each Cloud provider should be explored.

If we geographically distribute the Agents, we can also take into account latency constraints to improve the quality of the communications connecting each client to the closer Agent. Furthermore, if the Agents can be connected between them using trees we can improve the latencies even more. As we can see in [22] using hybrid clouds also offers cost saves in many cases.

Other interesting line is the design of algorithms with feedback that are modified in real-time taking into account the status of the Agents. Finally and putting all of these things together, it is interesting the study of common characteristics of videoconferencing deployments that allow us to generalise the requirements and design a global procedure to schedule the resources.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf [retrieved: March, 2013], 2009.

[2] Adobe Flash Player. [Online]. Available: http://get.adobe.com/en/flashplayer/ (retrieved: January, 2016)

[3] HTML5 W3C. [Online]. Available: http://dev.w3.org/html5/spec/ (retrieved: January, 2016)

[4] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "Webrtc 1.0: Real-time communication between browsers," August 2012.

[5] A. Alonso, P. Rodriguez, J. Salvachua, and J. Cerviño, "Deploying a multipoint control unit in the cloud: Opportunities and challenges," in CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, 2013, pp. 173–178.

[6] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu, "Independent tasks scheduling based on genetic algorithm in cloud computing," in Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on, Sept 2009, pp. 1–4.

[7] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, April 2010, pp. 400–407.

[8] X. Lu and Z. Gu, "A load-adapative cloud resource scheduling model based on ant colony algorithm," in Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, Sept 2011, pp. 296–300.

[9] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in Chinagrid Conference (ChinaGrid), 2011 Sixth Annual, Aug 2011, pp. 3–9.

[10] P. Rodriguez, A. Alonso, J. Salvachua, and J. Cervino, "dOTM: A mechanism for distributing centralized multi-party video conferencing in the cloud," in The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014). IEEE, 2014, pp. 61–67.

[11] P. Rodríguez, A. Alonso, J. Salvachúa, and J. Cervino, "Materialising a new architecture for a distributed mcu in the cloud," Computer Standards & Interfaces, pp. –, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548915001014 (retrieved: January, 2016)

[12] Licode. [Online]. Available: http://lynckia.com/licode (retrieved: January, 2016)

[13] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Requests for Comments, RFC Editor, RFC 5245, April 2010. [Online]. Available: http://tools.ietf.org/html/rfc5245 (retrieved: January, 2016)

[14] M. Baugher, "The Secure Real-time Transport Protocol (SRTP)," Internet Requests for Comments, RFC Editor, RFC 3711, March 2004. [Online]. Available: http://tools.ietf.org/html/rfc3711 (retrieved: January, 2016)

[15] E. Rescorla, "Datagram Transport Layer Security," Internet Requests for Comments, RFC Editor, RFC 4347, April 2006. [Online]. Available: http://tools.ietf.org/html/rfc4347 (retrieved: January, 2016)

[16] P. Rodríguez, D. Gallego, J. Cerviño, F. Escribano, J. Quemada, *et al.*, "Vaas: Videoconference as a service," in Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on. IEEE, 2009, pp. 1–11.

[17] "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0," OASIS Standard, Tech. Rep., October 2012.

[18] Openstack. [Online]. Available: http://www.openstack.org/software/openstack-compute/ (retrieved: January, 2016)

[19] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2 (retrieved: January, 2016)

[20] Google Compute Engine. [Online]. Available: http://cloud.google.com/compute (retrieved: January, 2016)

[21] Chromium. [Online]. Available: http://www.chromium.org/ (retrieved: January, 2016)

[22] J. Cervino, P. Rodriguez, I. Trajkovska, F. Escribano, and J. Salvachua, "A cost-effective methodology applied to videoconference services over hybrid clouds," Mobile Networks and Applications, vol. 18, no. 1, pp. 103–109, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11036-012-0380-4 (retrieved: January, 2016)

 ISBN: 978-1-61208-460-2